MOBILERL: ADVANCING MOBILE USE AGENTS WITH ADAPTIVE ONLINE REINFORCEMENT LEARNING

Yifan Xu[1†*], **Xiao Liu**[1,2*], **Xinghan Liu**[1†], **Jiaqi Fu**[1†], **Hanchen Zhang**[1†], **Bohao Jing**[2†], **Shudan Zhang**[1†], **Yuting Wang**[2], **Wenyi Zhao**[2], **Yuxiao Dong**[1]

[1] Tsinghua University [2] Zhipu AI

ABSTRACT

Vision language models (VLMs) have recently shown potential as general-purpose agents for graphical user interface (GUI) interaction. However, extending them to mobile environments remains challenging due to (i) complex instruction following under limited supervision, (ii) the heavy-tailed distribution of task difficulty, and (iii) the inefficiency of large-scale environment sampling. We introduce **MOBILERL**, a unified framework designed to enhance vision language agents in mobile GUI tasks. First, *Iterative Reasoning Refinement* converts action-only demonstrations into reasoning–action pairs via an off-the-shelf model and supervised fine-tuning, enabling more effective use of expert data. Second, we propose *Difficulty–Adaptive GRPO (DGRPO)*, an online reinforcement learning algorithm that integrates Difficulty-Adaptive Positive Replay, Failure Curriculum Filtering, and Shortest-Path Reward Adjustment, which jointly stabilize training by adapting to task difficulty and rewarding efficient solutions aligned with user preferences. We evaluate our approach by applying MOBILERL to GLM-4.1V-9B-Base (Team et al., 2025), resulting in AUTOGLM-Mobile-9B, which achieves state-of-the-art (SOTA) results with success rates of **75.8%** on AndroidWorld (Lai et al., 2025) and **46.8%** on AndroidLab (Xu et al., 2024). The algorithm and framework are adopted in building AUTOGLM (Liu et al., 2024)

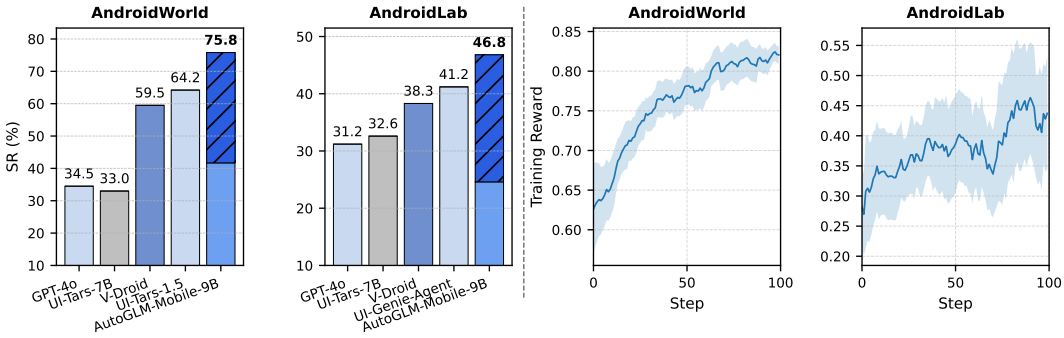


Figure 1: **Left:** Success rates (SR) on AndroidWorld (Rawles et al., 2024) and AndroidLab (Xu et al., 2024); hatched areas indicate gains from MOBILERL. **Right:** Trajectory-level rewards of MOBILERL with 95% CIs on training sets, showing consistent performance growth. Overall, MOBILERL outperforms prior work on both AndroidWorld (w/ rule-based rewards) and AndroidLab (w/ a reward model).

1 INTRODUCTION

Advanced vision language models (VLMs) have recently demonstrated promising capabilities as GUI agents (Bai et al., 2025; Team et al., 2025), enabling zero-shot interaction with web pages and mobile interfaces. Prior work (Rawles et al., 2023a; Xu et al., 2024; Bai et al., 2024; Lu et al., 2025) has used supervised fine-tuning or offline imitation learning GUI agents, which rely on static

*YX and XL contributed equally.

†Work done while these authors interned at Zhipu AI.

expert demonstrations to teach individual action mappings, but suffer from compounding errors when deployed in novel environments. The success of reasoning-model training has popularized the reinforcement learning with verifiable rewards paradigm (DeepSeek-AI et al., 2025). Single-step expert datasets naturally provide canonical action labels, rendering single-step reinforcement learning with expert demonstrations both straightforward and practical (Lu et al., 2025), but inherently limited by the inability to interact and train over full action sequences. In the GUI Agent domain, there have also been some explorations into online learning (Qi et al.; Dong et al., 2025; Dai et al., 2025); however, these directions still leave open how to systematically reduce exploration cost in mobile simulators.

Training mobile GUI agents that are robust and efficient in interactive mobile environments remains challenging for three practical reasons. (i) *Complex instruction following under sparse positive signals*: untuned base models struggle to stably produce action commands in the required format for complex, GUI-specific instructions (Xu et al., 2024), and the heavy cost and latency of mobile emulation make such correctly executed rollouts rare, leading to data-inefficient early exploration; the heavy cost and latency of mobil emulation make successful rollouts rare, rendering early exploration data-inefficient. (ii) *Large and unstable difficulty spectrum*: many tasks require multiple rollouts to succeed, while others are persistently unsolvable—naive sampling wastes budget and under-utilizes scarce but informative difficult successes. (iii) *Large-scale mobile environment sampling bottlenecks*: deploying and managing hundreds of concurrent mobile instances is resource-intensive, difficult to reproduce across setups, and often yields low sampling throughput, limiting the scale and efficiency of online reinforcement learning.

Motivated by these challenges, we introduce **MOBILERL**, a unified framework that integrates *warm-up stages* with *large-scale online reinforcement learning* for mobile GUI agents. During the warm-up phase, the base model is fine-tuned on expert demonstration data. A central difficulty lies in the fact that mobile-use expert demonstrations typically contain only the final action sequences, while omitting intermediate reasoning steps. To address this issue, we propose Iterative Reasoning Refinement, which leverages an off-the-shelf Instruct model to augment raw demonstrations with reasoning–action pairs, followed by supervised fine-tuning. This design fulfills two key objectives. First, by incorporating the explicit intermediate reasoning part, the model is able to acquire the ability to follow long and compositional instructions for the mobile agent more effectively, while simultaneously reducing the number of costly on-policy trials required to obtain positive rewards in mobile simulators. Second, it enables the full utilization of large-scale open-source or human-annotated expert datasets without dependence on proprietary models.

Our online reinforcement learning stage introduces **Difficulty–Adaptive Group Relative Policy Optimization (DGRPO)**, an extension of Group Relative Policy Optimization (GRPO) (Shao et al., 2024) that adapts optimization to instance difficulty and explicitly rewards solution efficiency. DGRPO incorporates three additional mechanisms. (1) *Difficulty-Adaptive Positive Replay (DAPR)* maintains a curated buffer of challenging, high-quality trajectories and balances them with fresh on-policy samples. In sparse-reward mobile environments, difficult successes are rare yet highly informative; replaying them amplifies their learning signal and stabilizes policy updates. (2) *Failure Curriculum Filtering (FCF)* down-weights persistently unsolvable tasks using online difficulty statistics, reallocating computational budget toward challenging but feasible instances. Given the heavy-tailed difficulty distribution observed in mobile phone use benchmarks (Xu et al., 2024; Rawles et al., 2024), pruning hard dead-ends improves sample efficiency while retaining signal from recoverable failures. (3) *Shortest-Path Reward Adjustment (SPA)* reshapes the reward function based on completion length, granting higher returns to shorter solutions. Length-sensitive rewards counteract the bias toward verbose and better align with user preferences in mobile interaction contexts.

To sustain high throughput, we employ a distributed sampling framework in which a gRPC controller orchestrates hundreds of dockerized Android virtual devices (AVDs) across multiple machines. This setup enables concurrent interaction with over 1000 environments while preserving reproducibility. Since most existing open-source benchmarks and simulators are built upon the Android operating system (Toyama et al., 2021; Rawles et al., 2024), this design ensures seamless compatibility and faithful reproduction of environment behaviors.

We build our pipeline on Qwen2.5-VL-7B-Instruct (Bai et al., 2025) and GLM-4.1V-9B-Base (Team et al., 2025), applying the proposed MOBILERL training framework to both backbones. For a direct

comparison under the same Qwen2.5-VL backbone (**7B variant**, which is significantly smaller than the 72B models used by UI-Tars-1.5 (Qin et al., 2025) and UI-Genie-Agent Xiao et al. (2025)), MOBILERL w/ Qwen2.5-VL-7B achieves substantially higher performance. Specifically, UI-Tars-1.5 attains 64.2% on ANDROIDWORLD and UI-Genie-Agent achieves 41.2% on ANDROIDLAB, whereas MOBILERL w/ Qwen2.5-VL-7B reaches 72.0% on ANDROIDWORLD and 42.5% on ANDROIDLAB, yielding absolute improvements of $+7.8$ and $+1.3$ points, respectively. Furthermore, when adopting the GLM-4.1V-9B backbone, MOBILERL w/ GLM-4.1V-9B-Base further improves to 75.8% on ANDROIDWORLD and 46.8% on ANDROIDLAB, establishing new state-of-the-art results with absolute gains of $+11.6$ and $+5.6$ points over the strongest previous baselines. We refer to this variant as AUTOGLM-Mobile-9B.

In summary, our contributions are as follows:

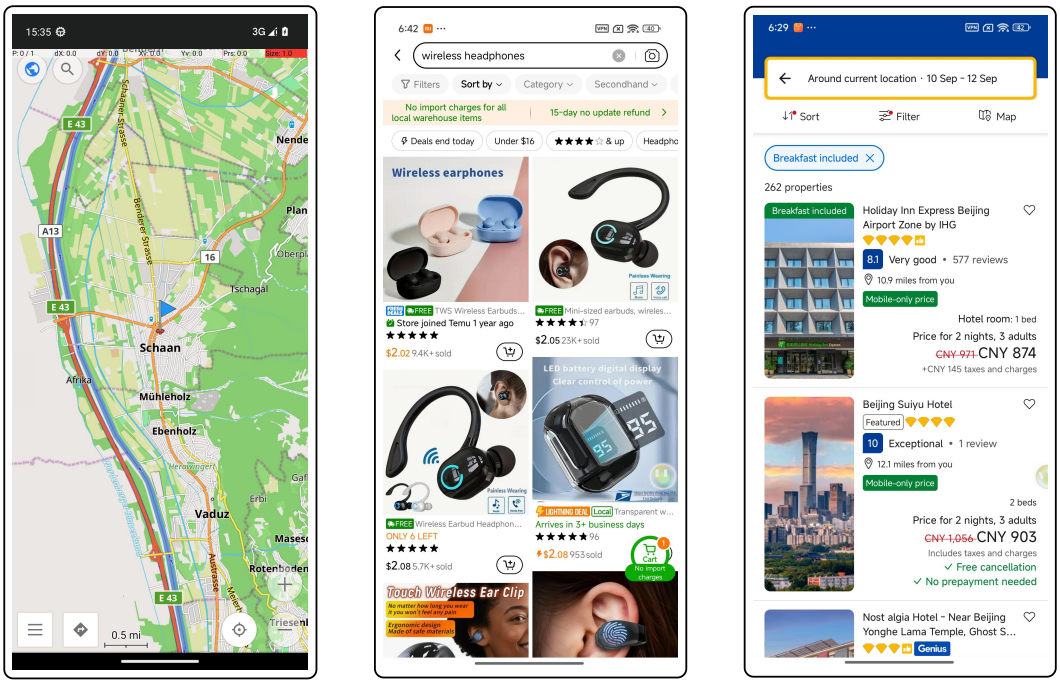
- **MOBILERL Framework & scalable sampling**: We propose MOBILERL, a framework that combines an iterative reasoning-augmented warm-up with online reinforcement learning for mobile GUI agents. We further establish a distributed sampling implementation that coordinates hundreds of dockerized Android virtual devices via gRPC, lowering early exploration cost, improving instruction adherence, and enabling reproducible large-scale training on Android benchmarks.
- **DGRPO Algorithm**: We introduce Difficulty–Adaptive Group Relative Policy Optimization (DGRPO), which extends GRPO with (i) *DAPR* for replaying challenging successful trajectories, (ii) *FCF* for down-weighting persistently unsolved tasks, and (iii) *SPA* for length-sensitive reward shaping, thereby accounting for instance difficulty and solution efficiency.
- **Empirical Results**: We demonstrate the effectiveness of our approach on ANDROIDWORLD and ANDROIDLAB. Using Qwen2.5-VL-7B-Instruct as the base model, MOBILERL w/ Qwen2.5-VL-7B achieves 72.0% and 42.5% , respectively. With GLM-4.1V-9B-Base under the same pipeline, AUTOGLM-Mobile-9B attains **75.8%** on ANDROIDWORLD and **46.8%** on ANDROIDLAB. We further integrate our pipeline into the AUTOGLM product.

2 RELATED WORK

2.1 MOBILE GUI AGENTS

Building on solid foundations of powerful language models, researchers design agents capable of controlling the interfaces of real devices (Agashe et al., 2025; Qin et al., 2025; Lai et al., 2025), including personal computers and mobile phones. Among these, the Mobile GUI agent interacts with Android devices by perceiving the graphical user interface (GUI) and executing actions such as tapping, swiping, and typing text (Toyama et al., 2021; Xu et al., 2024).

Considerable effort has been devoted to developing carefully designed frameworks and workflows to enhance agents' capability of predicting the next action and learning ability in mobile environments. Some works have proposed novel methods, such as utilizing a multimodal framework to enable agents to operate by exploration (Yang et al., 2023), utilizing a separate modules to enhance reasoning capabilities and generalization across tasks (Lai et al., 2025), utilizing a verifier-driven agent framework (Dai et al., 2025) and utilizing small language models coding capabilities to perform code-based execution (Wen et al., 2025). In addition to these, numerous approaches have also been presented, offering efficient behavior cloning and reinforcement learning based training methods. However, most of these approaches are still constrained to either offline reinforcement learning settings or single-turn interactions. For example, DigiRL (Bai et al., 2024) focuses on offline reinforcement learning using static human demonstration data, which limits the agent's ability to explore or adapt dynamically. U1-R1 (Lu et al., 2025) relies on single-turn data, where each episode contains only a single GUI action, preventing the agent from acquiring long-term planning or multi-step reasoning abilities. UI-Tars (Qin et al., 2025) adopts direct preference optimization (DPO), which, while effective for learning stepwise preferences, still operates in an offline regime and lacks true multi-turn interactive training. As a result, the potential of online or multi-turn reinforcement learning for interactive, adaptive mobile GUI agents remains largely underexplored.



(a) Add a location marker for 47.16, 9.51 in the OsmAnd maps app.

(b) Search for wireless headphones in temu, and sort by price low to high.

(c) Search for hotels on Booking, check-in date is 09-10, check-out date is 09-12, sort by prices.

Figure 2: Example mobile tasks finished by our agent. Our agent can automatically carry out tasks according to human instructions in both academic benchmarks and real-world applications.

2.2 Benchmarks for Mobile Agents

With the advancement of mobile GUI agents, continuous efforts have been made to design diverse benchmarks that provide comprehensive approaches to evaluate mobile GUI agents. Works like AndroidControl (Li et al., 2024) and Android in the Wild (Rawles et al., 2023b), along with MobileAgentBench (Wang et al., 2024) and Mobile-Bench (Deng et al., 2024), evaluate agents in a static or pre-recorded Android environment without supporting real-time online interaction. Although these benchmarks contain many tasks, their static simulation or offline replay of Android devices makes them unable to reflect real-world conditions. In contrast, environments such as AndroidWorld (Rawles et al., 2024), AndroidLab (Xu et al., 2024), and B-MOCA (Lee et al., 2024) utilize interactive Android emulator environments that cover a wide spectrum of apps and real-world tasks, yet they remain challenging for existing mobile agents. Notably, to the best of our knowledge, all publicly available mobile GUI agent benchmarks to date are limited to the Android operating system.

3 Problem Setting and Preliminaries

3.1 Problem Setting

We model device control and GUI navigation under natural-language instructions as a finite-horizon Markov Decision Process (MDP) (Littman, 2009), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, H)$. Here, $\mathcal{S}$ is the device state; $\mathcal{A}$ is a discrete action space covering `Tap`, `Swipe`, `Type`, `Long Press`, and `Finish`; $P(s_{t+1} \mid s_t, a_t)$ denotes environment dynamics induced by Android operating system and applications; $R(s_t, a_t)$ is a sparse reward that yields 1 iff the task is completed at step T and 0 otherwise; and H is the interaction horizon. Given an initial state–instruction pair $(s_0, c) \sim \mu_0$, an episode forms a trajectory $\tau = (s_0, a_0, \ldots, s_T)$ and terminates when the task is fulfilled or the horizon H is reached.

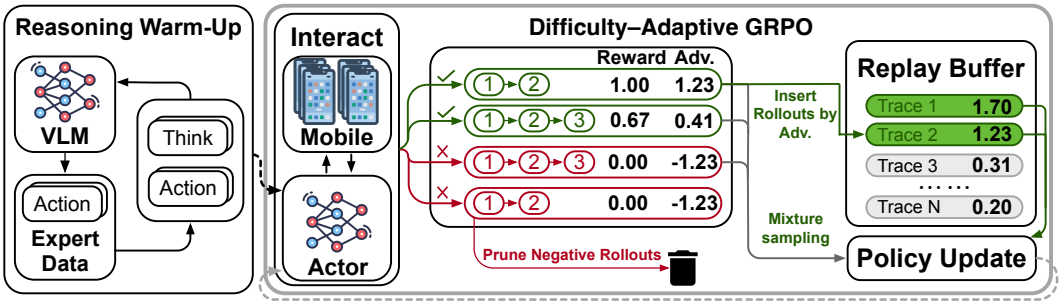


Figure 3: Overview of MOBILERL. (1) *Iterative Reasoning Refinement:* the model performs multi-round, self-bootstrapped rollouts on expert data to construct a reasoning-augmented initializer. (2) *Online training with DGRPO:* the warmed-up policy is refined via interaction; top positive trajectories enter a replay buffer while part of the negatives are pruned. Policy updates samples from a mixture of on-policy data and the buffer.

3.2 GRPO

Group Relative Policy Optimization (GRPO) (Shao et al., 2024) extends Proximal Policy Optimization by replacing the learned value baseline with an on-the-fly, group-relative baseline computed from a set of trajectories for the same task. This design removes the need for an extra critic while retaining the stability benefits of advantage normalization. For a state s, we sample a group of G actions $a_1, \ldots, a_G$ from the old policy $\pi_{\theta_{\text{old}}}$, and compute the group-relative advantage as:

$$A^{\text{GRPO}}(s, a_i) \;=\; \frac{r(s, a_i) - \text{avg}_{j=1}^{G}\, r(s, a_j)}{\text{std}_{j=1}^{G}\, r(s, a_j)},$$

where avg and std denote the sample mean and standard deviation over the group rewards $\{r(s, a_j)\}_{j=1}^{G}$.

The policy maximizes the clipped surrogate

$$L^{\text{GRPO}}(\theta) = \mathbb{E}_{s, a_{1:G} \sim \pi_{\theta_{\text{old}}}} \Bigg[\underbrace{\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \min\Big(r_{i,t}(\theta)\, \hat{A}_{i,t},\; \text{clip}\big(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon\big)\, \hat{A}_{i,t} \Big)}_{\text{Clipped group surrogate}}$$

$$\underbrace{- \beta\, D_{\text{KL}}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big)}_{\text{KL penalty}} \Bigg],$$

where $r_{i,t}(\theta) = \frac{\pi_\theta(a_{i,t}|s)}{\pi_{\theta_{\text{old}}}(a_{i,t}|s)}$ and $D_{\text{KL}}(p\|q) = \sum_a p(a) \log \frac{p(a)}{q(a)}$.

4 MOBILERL

4.1 OVERVIEW

The MOBILERL framework comprises three components: supervised fine-tuning on expert demonstration data, an iterative warm-up stage *Iterative Reasoning Refinement*, and *Difficulty–Adaptive GRPO* (DGRPO). Because sampling in virtual-device environments is inefficient, starting online reinforcement learning directly from a base model was found to be excessively time-consuming in preliminary experiments. We begin by following the data collection protocol of Xu et al. (2024) to obtain expert demonstrations, which are then used for supervised fine-tuning. We then construct a stronger reasoning initializer via Iterative Reasoning Refinement over the expert dataset, followed by the application of DGRPO for efficient online refinement.

For the observation space, we adopt a dual presentation: the current screenshot and a compressed Extensible Markup Language (XML). The preprocessing steps for XML simplification are detailed

in Appendix D. In most cases, the agent can use coordinates from XML to specify click positions, bypassing brittle pixel-level grounding; when graphical cues or incomplete XML are involved, the screenshot provides the necessary visual details.

4.2 ITERATIVE REASONING REFINEMENT (IRR)

Manually collected expert demonstrations dataset for mobile use often contains only the final action sequence, omitting intermediate reasoning. Training solely on such "black-box" trajectories yields opaque policies, while many unlabeled tasks remain unused. We leverage an off-the-shelf Instruct model to activate expert data and bootstrap a reasoning-augmented training set from raw demonstrations, yielding a structured and transparent policy initialization.

Concretely, we iteratively build reasoning instruction–tuning pairs in three stages:

- **Stage 0 (Bootstrap sampling):** For each task x with expert answer a^*, the Instruct model M generates diverse candidate reasoning–action pairs (c_k, a_k) (via, e.g., temperature/nucleus sampling). Whenever $a_k = a^*$, we retain (x, c_k, a^*) in $\mathcal{D}_R$.
- **Stage 1 (Supervised fine-tuning):** Train an initial reasoning policy π_0^R on $\mathcal{D}_R$.
- **Stage 2 (Iterative refinement):** At iteration t, π_t^R proposes candidates; those matching a^* are scored by conciseness and $\log P(c)$. The best explanation c^* is added to $\mathcal{D}_{\text{new}}$, and π_{t+1}^R is obtained by fine-tuning. We stop when the match rate saturates.

4.3 DIFFICULTY–ADAPTIVE GRPO(DGRPO)

Building on GRPO, we propose **D**ifficulty–Adaptive GRPO (DGRPO) tailored to our tasks. In multi-turn mobile use tasks, where immediate rewards are not available within a single round as in single-turn tasks (Lu et al., 2025; Team et al., 2025), the reward allocation strategy must be carefully redesigned. Beyond simply assigning a uniform final reward at the trajectory level, we introduce the *Shortest-Path Reward Adjustment*, which reshapes rewards with respect to task length. This adjustment provides more informative learning signals, guiding the model toward both accurate and efficient completion paths, while also facilitating the computation of trajectory-level advantages. This also facilitates the computation of trajectory-level advantages tailored to sparse-reward settings.

A further challenge arises from the uniform sampling strategy employed in standard GRPO. In mobile use scenarios—where each sample carries a significant computational cost—this approach results in poor sample efficiency, particularly due to the repeated inclusion of inherently unsolvable tasks. To mitigate this, we adapt data collection and training based on instance difficulty through two mechanisms: *Difficulty–Adaptive Positive Replay* and *Failure Curriculum Filtering*. At the same time, we restrict redundant successful trajectories to avoid unnecessary updates and promote training efficiency.

4.3.1 SHORTEST-PATH REWARD ADJUSTMENT (SPA).

In mobile tasks, the environment returns a binary terminal reward $r \in \{0, 1\}$ (Xu et al., 2024; Rawles et al., 2024) indicating task success. Previous approaches typically broadcast this reward to every timestep, i.e., $R(s_t, a_t) = r,\ t = 0, \ldots, T$, so that the per-step signal remains aligned with the sparse objective. However, assigning identical rewards to all successful rollouts biases training toward *longer* trajectories, since they contribute more gradient terms. To counteract this, we introduce SPA, which re-scales the reward for each trajectory τ_i as

$$R^{\text{SPA}}(s_t, a_t) = \begin{cases} 1 - \alpha \dfrac{T_i - T_{\min}}{T_i}, & r(\tau_i) = 1, \\ 0, & r(\tau_i) = 0, \end{cases} \qquad T_{\min} = \min_{\tau_j \in \mathcal{T}_{\text{succ}}} |\tau_j|,\ \alpha \in (0, 1], \quad (1)$$

where $T_i = |\tau_i|$ is the length of trajectory τ_i, and $\mathcal{T}_{\text{succ}} = \{\tau_j \mid r(\tau_j) = 1\}$ denotes the set of *successful* trajectories for the current problem instance. Here $T_{\min}$ is the length of the shortest successful trajectory in $\mathcal{T}_{\text{succ}}$, and $\alpha \in (0, 1]$ controls the penalty strength. In this formulation, shorter sequences are not automatically considered better; unsuccessful early terminations still receive a reward of 0. This adjustment encourages the policy to prefer shorter, successful paths without sacrificing the success rate.

Trajectory-level Advantage. To evaluate the contribution of each trajectory during training, we define a *trajectory-level advantage* that integrates the normalized rewards across the entire episode. Unlike step-wise advantage estimates common in actor–critic methods, this trajectory-level formulation aligns better with sparse-reward tasks, where reward signals are only available at the end of trajectories. Given a task, we sample a group of G trajectories $\mathcal{G} = \{\tau_1, \ldots, \tau_G\}$. For trajectory τ_i with episode length T_i, let $r_t^{\text{spa}}(\tau_i)$ denote the SPA reward at step t. We use the *step-wise* group mean (used as a baseline), then aggregate the deviations from this baseline to obtain a trajectory score for τ_i:

$$S_i = \sum_{t=1}^{T_i} \left(r_t^{\text{spa}}(\tau_i) - \frac{1}{G} \sum_{j=1}^{G} r_t^{\text{spa}}(\tau_j) \right). \tag{2}$$

Finally, we normalize these scores within the group to form the trajectory-level advantage:

$$A_{\text{traj}}(\tau_i) = \frac{S_i - \text{avg}_{j=1}^{G} S_j}{\text{std}_{j=1}^{G} S_j} \tag{3}$$

Unless otherwise stated, the normalization statistics avg and std are computed *within the group* over the G trajectories.

4.3.2 Difficulty–Adaptive Positive Replay (DAPR)

Buffer construction. At iteration t, the rollout set is $\mathcal{T}_t = \{\tau^{(1)}, \ldots, \tau^{(N)}\}$, collected under the current policy π_{θ_t}. We compute $A_{\text{traj}}(\tau)$ via equation 3 and insert the top κ trajectories into the replay buffer $\mathcal{B}$.

Mixture sampling. Each policy update is performed on a mini-batch of M trajectories obtained from the mixture distribution

$$q(\tau) \;=\; \gamma\, p_{\mathcal{B}}(\tau) \;+\; (1-\gamma)\, p_{\text{on}}(\tau) \tag{4}$$

where p_{on} is the on-policy distribution π_{θ_t} and $p_{\mathcal{B}}$ is the empirical distribution over $\mathcal{B}$. To keep the replay contribution under control, at most γM trajectories with the highest *current* advantage are drawn from $\mathcal{B}$, preserving on-policy diversity.

Pruning negative rollouts. To further stabilize training, we selectively prune trajectories with the lowest advantages to reduce the probability of introducing noisy samples into the replay buffer. Since high-advantage trajectories, if stored, are expected to be additionally sampled only once on average, we enforce a positive-to-negative trajectory ratio of at most 1:2 by discarding the lowest-advantage trajectories in descending order of trajectory-level Kullback–Leibler (KL) divergence. The KL divergence is computed between the current policy π_{θ_t} and the behavior policy that generated the trajectory τ.

DGRPO loss. Define the token-level importance ratio

$$r(\tau;\theta) = \prod_{t=0}^{T} \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \quad \tilde{r}(\tau;\theta) = \text{clip}\big(r(\tau;\theta),\, 1-\epsilon,\, 1+\epsilon\big).$$

Here, $\pi_{\theta_{\text{old}}}$ denotes the behavior policy used to collect trajectories τ, serving as the reference distribution in importance sampling. With mixture sampling $q(\tau)$ and trajectory-level advantage $A_{\text{traj}}(\tau)$ from equation 3–equation 4, the modified GRPO objective becomes

$$\mathcal{L}_{\text{DGRPO}}(\theta) = -\,\mathbb{E}_{\tau\sim q}\Big[\min\big(r(\tau;\theta)\, A_{\text{traj}}(\tau),\; \tilde{r}(\tau;\theta)\, A_{\text{traj}}(\tau)\big)\Big] \;+\; \beta\, D_{\text{KL}}\big[\pi_\theta \,\big\|\, \pi_{\theta_{\text{ref}}}\big],$$

where ϵ is the clipping hyper-parameter and β weights the KL regulariser.

4.3.3 Failure Curriculum Filtering (FCF)

To avoid repeatedly sampling tasks that yield zero reward—which wastes computation and hinders the collection of positive advantage data—we propose Failure Curriculum Filtering (FCF). In FCF, any task producing all-zero rewards for two consecutive epochs enters a three-epoch cooldown, during which its sampling probability is reduced according to $w_{\text{task}} = \exp(-f)$, where f is the number of consecutive failure epochs. Tasks with $f \geq 3$ are permanently removed from the sampling pool. For stability, failure histories from previous training are retained.

Table 1: Action Space for Mobile GUI Interaction. We utilize the action space from AndroidLab (Xu et al., 2024), which represents screen positions with bounding boxes aligned to XML data. Unlike other works (Qin et al., 2025; Rawles et al., 2024), we exclude the "press home" action due to its variability across device models.

Action	Parameters	Description
Tap	`element=[x1,y1,x2,y2]`	Tap at the rectangle defined by top-left (x1,y1) and bottom-right (x2,y2).
Type	`text={string}`	Enter the given string into the focused input field.

Swipe	`direction={up/down/left/right}`
	`dist={short/medium/long}`
	`element=[x1,y1,x2,y2] (optional)`
		Swipe in the given direction over the specified distance. Optionally constrain to the rectangle element.

Long Press	`element=[x1,y1,x2,y2]`	Press and hold on the given rectangle area.
Launch	`app={AppName}`	Launch the named application.
Back	none	Press the system Back button.
Finish	`message={string} (optional)`	End the session with an optional message.

4.4 Training

The overall training framework consists of three stages:

- **Supervised fine-tuning (SFT):** We curate an instruction-tuning expert dataset covering more than 50 applications and 500k interaction steps. This dataset is used to perform reasoning-free supervised fine-tuning of the base model.
- **Iterative Reasoning Refinement (IRR):** Building upon the supervised stage, we employ the Iterative Reasoning Refinement procedure on the expert dataset to construct a reasoning-oriented fine-tuning corpus. Training on this corpus for an additional 60k steps yields a reasoning warm-up model.
- **DGRPO:** In the final stage, we perform online sampling from a curated suite of tasks to optimize agent behavior.

For fair comparison with the current state-of-the-art UI-Tars-1.5 (Qin et al., 2025), we adopt Qwen2.5-VL-7B-Instruct (Bai et al., 2025) as our initialization model. In addition, we apply the same training framework to GLM-4.1V-9B-Base (Team et al., 2025), resulting in **AutoGLM-Mobile-9B**.

5 Experiments

5.1 Experiments Settings

Datasets and Benchmarks To comprehensively evaluate our method and the baselines, we utilize two benchmarks: AndroidWorld (Rawles et al., 2024), which consists of 116 tasks across 20 applications, and AndroidLab (Xu et al., 2024), which consists of 138 tasks across 9 applications. Together, these benchmarks provide a total of 254 tasks covering diverse real-world scenarios and goals. Both benchmarks offer interactive environments where agents execute actions to complete tasks from natural language instructions, with trajectories evaluated by specific rules.

For reinforcement learning, we construct extended training splits from both benchmarks by varying initial states, task requirements, and task compositions, while ensuring that any overlap with the evaluation sets is excluded. The resulting training corpus comprises the full training split of AndroidWorld and approximately 1k tasks drawn from AndroidLab, with mixed sampling applied at a fixed ratio of 4:1 (AndroidWorld: AndroidLab). For AndroidWorld, we directly adopt its rule-based trajectory rewards, whereas for AndroidLab, which does not provide training rewards, we employ a

Table 2: Success rates (SR, %) of proprietary and open-source models on AndroidWorld and AndroidLab benchmarks for mobile GUI interaction tasks. AUTOGLM-Mobile-9B achieves the highest performance on both benchmarks.

Models	AndroidWorld (SR)	AndroidLab (SR)
Proprietary Models		
GPT-4o-2024-11-20 (OpenAI, 2023)	34.5	31.2
Claude-Sonnet-4-20250514-thinking (Anthropic, 2023)	41.0	40.6
UI-Tars-1.5 (Qin et al., 2025)	64.2	38.3
AUTOGLM-2024-10 (Liu et al., 2024)	–	36.2
Open Models		
Qwen2.5-VL-7B-Instruct (Bai et al., 2025)	27.6	10.1
GLM-4.1V-9B-Thinking (Team et al., 2025)	41.7	24.6
UI-Tars-7B (Qin et al., 2025)	33.0	32.6
V-Droid (Dai et al., 2025)	59.5	38.3
UI-Genie-Agent (Xiao et al., 2025)	-	41.2
MOBILERL		
- w/ Qwen2.5-VL-7B	72.0	42.5
- w/ GLM-4.1V-9B-Base (AUTOGLM-Mobile-9B)	**75.8**	**46.8**

VLM-based reward model to generate reward signals. Full details of the reward design are provided in Appendix C.

Training infrastructure. We implement our algorithm in an AgentRL-based framework (Zhang et al., 2025), use FSDP (Zhao et al., 2023) for training, and SGLang (Zheng et al., 2024) as the inference engine. We build an asynchronous sampling framework for Android virtual devices that supports large-scale, high-concurrency sampling. A gRPC + Protocol Buffers protocol enables parallel sampling across multiple machines. Each Android virtual device runs in a Docker container with only Android Debug Bridge (ADB) and gRPC ports exposed, allowing multiple Android virtual device instances per host and concurrent task handling.

Our experiments are conducted on a cluster of four machines, each equipped with eight GPUs. In addition to GPU resources, we employ a CPU cluster to support large-scale environment sampling. Each CPU node provides 120 physical cores (240 threads) and 1 TiB of memory. In practice, a CPU node can efficiently host nearly as many Android virtual devices as its number of physical cores.

Baselines Our baselines encompass both closed-source and open-source agents and models, including closed-source LLMs: GPT-4o-2024-11-20 (OpenAI, 2023) and Claude-Sonnet-4-20250514-thinking (Anthropic, 2023), closed-source agents UI-Tars-1.5 (Qin et al., 2025) and AutoGLM (Liu et al., 2024), as well as open-source VLMs, including Qwen2.5-VL-7B-Instruct (Bai et al., 2025), GLM-4.1V-9B-Thinking (Team et al., 2025), UI-Tars-7B (Qin et al., 2025), and V-Droid (Dai et al., 2025). A comprehensive comparison of their configurations and performance is detailed in the Results section.

Action Space We design a series of actions based on AndroidLab (Xu et al., 2024), which supports a concise set of actions for GUI interaction. Each action may take specific parameters, described in Table 1.

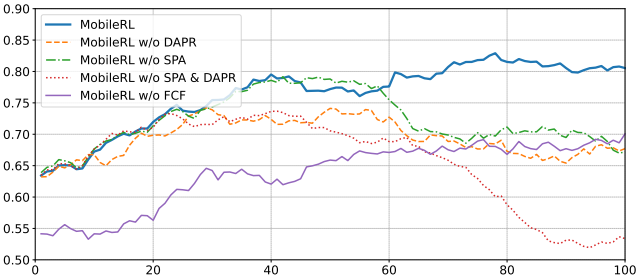
5.2 MAIN RESULTS

We evaluate MOBILERL with Qwen2.5-VL-7B and GLM-4.1V-9B-Base as backbones on the AndroidWorld and AndroidLab benchmarks. As shown in Table 2, our method substantially outperforms both proprietary (e.g., GPT-4o: 34.5% / 31.2%). With Qwen2.5-VL-7B as the backbone, attains 72.0% on AndroidWorld and 42.5% on AndroidLab, outperforming prior state-of-the-art approaches. With GLM-4.1V-9B as the backbone, performance further improves to 75.8% on AndroidWorld and 46.8% on AndroidLab—the highest across all models.

Table 3: Ablation study of MOBILERL framework. We incrementally apply SFT, IRR, and DGRPO. *For the GLM-4.1V-9B series models, training is based on GLM-4.1V-9B-Base, but we compare against GLM-4.1V-9B-Thinking since the base model cannot reasonably measure mobile agent scores.

Models	AndroidWorld (SR)	AndroidLab (SR)
Qwen2.5-VL-7B-Instruct (Bai et al., 2025)	27.6	10.1
+ SFT	50.2$_{+22.6}$	36.9$_{+26.8}$
+ IRR	56.8$_{+29.2}$	38.7$_{+28.6}$
+ DGRPO(MOBILERL)	72.0$_{+44.4}$	42.5$_{+32.4}$
GLM-4.1V-9B-Thinking* (Team et al., 2025)	41.7	24.6
+ SFT	48.9$_{+7.2}$	39.8$_{+15.2}$
+ IRR	66.1$_{+24.4}$	40.3$_{+15.7}$
+ DGRPO(MOBILERL)	75.8$_{+34.1}$	46.8$_{+22.2}$

5.3 ABLATION STUDY



(a) Trajectory-level reward (y-axis) from the environment on the training set concerning training steps (x-axis).

Model	AW (SR)
MOBILERL	**71.1**
w/o DAPR	63.6
w/o SPA	69.1
w/o DAPR&SPA	58.5
w/o FCF	64.8

(b) Test performance (SR, %) on the AndroidWorld test set.

Figure 4: Ablation study results: (a) Training trajectory-level rewards; (b) Test performance on AndroidWorld under different model variants. All models are trained solely on the AndroidWorld training set, and results are averaged over three runs to reduce the impact of stochasticity.

To assess the contributions of the MOBILERL framework and the individual components of the DGRPO algorithm, we perform an ablation study as presented in Table 3 and Figure 4. We begin with Qwen2.5-VL-7B-Instruct and GLM-4.1V-9B-Base as the base model and incrementally apply SFT, IRR, and DGRPO. Subsequently, taking the Qwen2.5-VL-7B-Instruct trained with SFT and IRR as the initial model, we conduct a detailed analysis of the impact of each constituent component of DGRPO.

MOBILERL Framework Ablation We conduct a sequential ablation study to evaluate the contribution of SFT, IRR, and DGRPO within the MOBILERL framework (Table 3). Across both Qwen2.5-VL-7B and GLM-4.1V-9B, we observe consistent and progressive improvements at each stage. SFT provides a strong initial performance boost, yielding average gains of around 20–25% across benchmarks, while IRR introduces additional improvements of approximately 5–10%. The final DGRPO stage achieves the highest overall success rates, delivering total gains of over 40% for Qwen2.5-VL and more than 30% for GLM compared to their respective baselines. These results highlight the complementary nature of the three components and demonstrate the effectiveness of combining supervised fine-tuning, iterative reasoning refinement, and reinforcement learning in enhancing agent capabilities.

DGRPO Algorithm Ablation To eliminate potential bias introduced by the AndroidLab reward model, we perform ablation experiments using only the AndroidWorld training data. Specifically, we examine the following variants of our method:

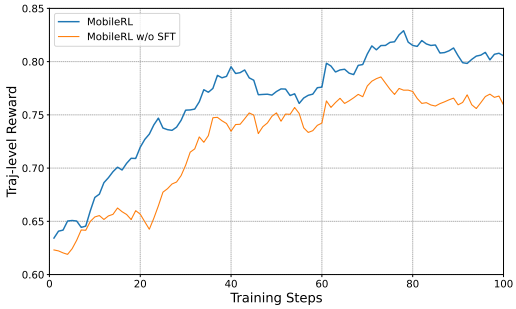


Figure 5: Comparison between direct training on Reasoning Warm-up with reinforcement learning and using reasoning-free SFT beforehand. The curves demonstrate the effectiveness of reasoning-free SFT.

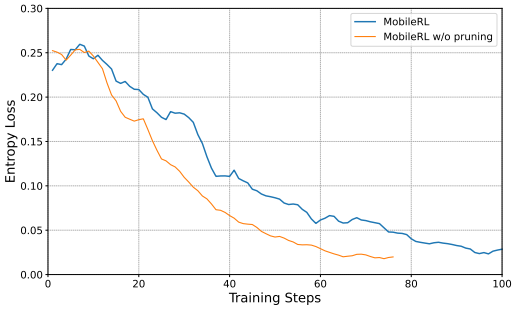


Figure 6: Effect of applying the pruning-negative strategy. Pruning consistently yields higher entropy loss, indicating improved exploration. Training without pruning is terminated early due to the lack of diverse data.

- **MOBILERL w/o DAPR**: training without using the replay buffer;
- **MOBILERL w/o SPA**: directly using the initial reward signal without reward shaping;
- **MOBILERL w/o DAPR & SPA**: disabling both the replay buffer and reward shaping;
- **MOBILERL w/o FCF**: uniformly sampling from the entire training set instead of filtering candidate functions.

We report two evaluation metrics: (1) the trajectory-level reward curves during training (excluding trajectories collected from the replay buffer); and (2) the final success rates (SR) on the AndroidWorld test set.

Based on the results in Figure 4, we make the following observations: (1) FCF is essential for data filtering—without it, early sampling is dominated by overly difficult tasks, resulting in too many negative examples and a lower overall reward ceiling despite later improvements. Specifically, disabling FCF reduces the test SR from 71.1 to 64.8. Thus, FCF is retained in the remaining ablation variants; (2) training with only FCF is initially stable but collapses after around 30 steps, consistent with the sharp performance drop observed in the w/o DAPR&SPA variant (final SR 58.5); (3) both SPA and DAPR contribute to performance gains. Removing SPA decreases SR from 71.1 to 69.1, while removing DAPR causes a larger drop to 63.6, showing that DAPR has a more pronounced effect.

5.4 IS REASONING-FREE SFT STILL NECESSARY?

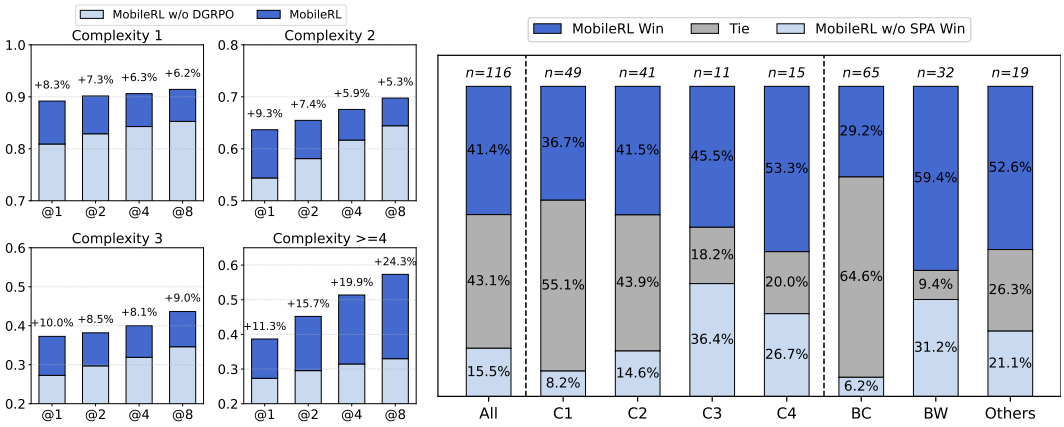
In this setting, we directly apply supervised fine-tuning (SFT) on the expert dataset without reasoning traces, which we term *Reasoning-Free SFT*. The fine-tuning data in this stage contains only action sequences, without intermediate thought processes. Due to training instability, we are unable to perform reinforcement learning directly on models without any reasoning-related SFT. This raises a key question: *Is fine-tuning with expert data that lacks reasoning still beneficial?*

We compare two configurations: (1) Reasoning-Free SFT followed by IRR and DGRPO; and (2) IRR and DGRPO without the Reasoning-Free SFT stage.

As shown in Figure 5, incorporating Reasoning-Free SFT consistently leads to higher performance across benchmarks, indicating that even without explicit reasoning traces, expert demonstrations contribute to stabilizing training and enhancing final outcomes.

5.5 EFFECT OF PRUNING NEGATIVE TRAJECTORIES

We further study a pruning strategy that discards overly frequent erroneous trajectories from the reinforcement learning buffer before each update. As depicted in Figure 6, this filtering keeps the policy entropy consistently higher during training. By pruning trajectories whose advantages remain



(a) Pass@k across complexity levels.

(b) Comparison of task completion steps.

Figure 7: Performance comparison on AndroidWorld test set. (a) Pass@k improvements across different complexity levels. Pass@k is computed as the proportion of tasks solved within the top-k attempts. (b) Win rate comparison between MOBILERL and MOBILERL w/o SPA, where *win* indicates completing the task with fewer steps. n denotes the number of task templates in each category. *All* represents all templates; *C1–C4* refer to complexity levels 1–4; *BC* (Both Correct) and *BW* (Both Wrong) indicate cases where both methods succeed or fail, respectively; *Others* covers cases where only one method is correct.

persistently negative, the agent avoids being driven by detrimental gradients; probability mass is instead spread over a broader action space, fostering exploration and delaying premature convergence.

5.6 SUCCESS RATES BY TASK COMPLEXITY

As illustrated in Figure 7a, we evaluated the improvement in model capabilities before and after reinforcement learning training by categorizing the results based on task difficulty. We conduct 8 test runs using a temperature of 1.0, and employ the AndroidWorld test set to assign difficulty levels based on the rounded-up value of Complexity (defined by Rawles et al. (2024)): a level of 1 indicates tasks solvable by humans within 10 steps, 2 within 20 steps, 3 within 30 steps, and ≥ 4 for tasks requiring more than 30 steps.

We measure performance using pass@1, 2, 4, and 8. Notably, our method achieves consistent improvements across all difficulty levels, with the magnitude of improvement increasing alongside task complexity. Furthermore, the post-reinforcement learning pass@1 scores exceeded the pre-reinforcement learning pass@8 scores, demonstrating the efficacy of our approach in unlocking the model's potential. Meanwhile, the relatively modest gains in pass@8 suggest that our strategy does not inhibit the model's ability to explore previously unsolved tasks, but rather constrains exploration within a more effective and reasonable range.

From another perspective, as task complexity increases—that is, for more complex and difficult tasks—the magnitude of improvement after training becomes larger. This also indicates that our solution, DAPR, which is specifically designed for heavy-tailed task difficulty, places greater emphasis on efficient training for higher-difficulty tasks (complexity = 3 or above, accounting for 22% of the total tasks).

5.7 IMPACT OF SPA ON STEP EFFICIENCY

Although the ablation study shows that the use of SPA has the least impact on the overall accuracy, we compare the completion lengths of MOBILERL and MOBILERL w/o SPA on the test set to demonstrate that SPA significantly reduces the number of steps required to complete a task. As shown in Figure 7b, we first divide the tasks according to their complexity levels and find that, across all difficulty levels, SPA shortens the required steps to varying degrees. We further categorize the tasks into three groups for a fairer comparison: cases where both models are correct (BC), both are wrong (BW), and others (only one model is correct). The results show that, in BC cases, MOBILERL

achieves shorter completions in 6.15% of tasks compared to 29.23% for MOBILERL w/o SPA; in BW cases, the rates are 31.25% vs. 59.38%; and in Others, 21.05% vs. 52.63%. These results confirm that SPA consistently improves step efficiency, even when both models succeed or fail.

6 CONCLUSION

In this work, we present MOBILERL, a framework for advancing mobile GUI agents by integrating staged initialization with an adaptive reinforcement learning algorithm. Training begins with supervised fine-tuning (SFT) on large-scale expert demonstrations to establish a strong task-oriented foundation. We then introduce a reasoning-augmented warm-up stage that enriches action sequences with intermediate rationales, improving transparency and reducing cold-start exploration cost.

Building on this initialization, we propose Difficulty–Adaptive GRPO (DGRPO), which extends GRPO with mechanisms tailored to sparse-reward, multi-turn settings: Shortest-Path Reward Adjustment for trajectory-length sensitivity, Difficulty–Adaptive Positive Replay for emphasizing high-advantage samples, and Failure Curriculum Filtering for mitigating unsolvable tasks. These components collectively improve sample efficiency and guide policies toward accurate and efficient task completion.

Experiments on AndroidWorld and AndroidLab show that MOBILERL achieves superior performance over both open-source and closed-source baselines, with ablation studies confirming the contributions of each stage.

REFERENCES

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents, 2025. URL https://arxiv.org/abs/2504.00906.

Anthropic. Introducing claude, 2023. URL https://www.anthropic.com/index/introducing-claude.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning, 2024. URL https://arxiv.org/abs/2406.11896.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Gaole Dai, Shiqi Jiang, Ting Cao, Yuanchun Li, Yuqing Yang, Rui Tan, Mo Li, and Lili Qiu. Advancing mobile gui agents: A verifier-driven approach to practical deployment, 2025. URL https://arxiv.org/abs/2503.15937.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Shihan Deng, Weikai Xu, Hongda Sun, Wei Liu, Tao Tan, Liujianfeng Li, Ang Li, Jian Luan, Bin Wang, Rui Yan, and Shuo Shang. Mobile-bench: An evaluation benchmark for LLM-based mobile agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Long Papers)*, pp. 8813–8831, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.478.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng

Dou. Agentic reinforced policy optimization, 2025. URL https://arxiv.org/abs/2507.19849.

Hanyu Lai, Junjie Gao, Xiao Liu, Yifan Xu, Shudan Zhang, Yuxiao Dong, and Jie Tang. Androidgen: Building an android language agent under data scarcity, 2025. URL https://arxiv.org/abs/2504.19298.

Juyong Lee, Taywon Min, Minyong An, Dongyoon Hahm, Haeone Lee, Changyeon Kim, and Kimin Lee. Benchmarking mobile device control agents across diverse configurations, 2024. URL https://arxiv.org/abs/2404.16660.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on computer control agents. *arXiv preprint arXiv:2406.03679*, 2024.

Michael L Littman. A tutorial on partially observable markov decision processes. *Journal of Mathematical Psychology*, 53(3):119–125, 2009.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, Junjie Gao, Junjun Shan, Kangning Liu, Shudan Zhang, Shuntian Yao, Siyi Cheng, Wentao Yao, Wenyi Zhao, Xinghan Liu, Xinyi Liu, Xinying Chen, Xinyue Yang, Yang Yang, Yifan Xu, Yu Yang, Yujia Wang, Yulin Xu, Zehan Qi, Yuxiao Dong, and Jie Tang. Autoglm: Autonomous foundation agents for guis, 2024. URL https://arxiv.org/abs/2411.00820.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanjing Xiong, and Hongsheng Li. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning, 2025. URL https://arxiv.org/abs/2503.21620.

OpenAI. Gpt-4 technical report, 2023.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations*.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents, 2025. URL https://arxiv.org/abs/2501.12326.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*, 2023a.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control, 2023b. URL https://arxiv.org/abs/2307.10088.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2024. URL https://arxiv.org/abs/2405.14573.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

GLM-V Team, Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihan Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, Baoxu Wang, Boyan Shi, Changyu Pang, Chenhui Zhang, Da Yin, Fan Yang, Guoqing Chen, Jiazheng Xu, Jiali Chen, Jing

Chen, Jinhao Chen, Jinghao Lin, Jinjiang Wang, Junjie Chen, Leqi Lei, Letian Gong, Leyi Pan, Mingzhi Zhang, Qinkai Zheng, Sheng Yang, Shi Zhong, Shiyu Huang, Shuyuan Zhao, Siyan Xue, Shangqin Tu, Shengbiao Meng, Tianshu Zhang, Tianwei Luo, Tianxiang Hao, Wenkai Li, Wei Jia, Xin Lyu, Xuancheng Huang, Yanling Wang, Yadong Xue, Yanfeng Wang, Yifan An, Yifan Du, Yiming Shi, Yiheng Huang, Yilin Niu, Yuan Wang, Yuanchang Yue, Yuchen Li, Yutao Zhang, Yuxuan Zhang, Zhanxiao Du, Zhenyu Hou, Zhao Xue, Zhengxiao Du, Zihan Wang, Peng Zhang, Debing Liu, Bin Xu, Juanzi Li, Minlie Huang, Yuxiao Dong, and Jie Tang. Glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning, 2025. URL https://arxiv.org/abs/2507.01006.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.

Luyuan Wang, Yongyu Deng, Yiwei Zha, Guodong Mao, Qinmin Wang, Tianchen Min, Wei Chen, and Shoufa Chen. Mobileagentbench: An efficient and user-friendly benchmark for mobile llm agents. 2024. arXiv preprint arXiv:2406.08184.

Hao Wen, Shizuo Tian, Borislav Pavlov, Wenjie Du, Yixuan Li, Ge Chang, Shanhui Zhao, Jiacheng Liu, Yunxin Liu, Ya-Qin Zhang, and Yuanchun Li. Autodroid-v2: Boosting slm-based gui agents via code generation, 2025. URL https://arxiv.org/abs/2412.18116.

Han Xiao, Guozhi Wang, Yuxiang Chai, Zimu Lu, Weifeng Lin, Hao He, Lue Fan, Liuyang Bian, Rui Hu, Liang Liu, Shuai Ren, Yafei Wen, Xiaoxin Chen, Aojun Zhou, and Hongsheng Li. Ui-genie: A self-improving approach for iteratively boosting mllm-based mobile gui agents, 2025. URL https://arxiv.org/abs/2505.21496.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android autonomous agents, 2024. URL https://arxiv.org/abs/2410.24024.

Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023.

Hanchen Zhang, Xiao Liu, Bowen Lv, Xueqiao Sun, Bohao Jing, Iat Long Iong, Zehan Qi, Hanyu Lai, Yifan Xu, Rui Lu, Zhenyu Hou, Hongning Wang, Jie Tang, and Yuxiao Dong. Agentrl: Reinforce all-round agents from zero. *arXiv preprint*, 2025.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023. URL https://arxiv.org/abs/2304.11277.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

A Case Study

We present a case study from **AndroidWorld** evaluating three agents: the SFT agent, the MobileRL w/o DGRPO agent, and the full MobileRL agent. The task, `SimpleCalendarAddRepeatingEvent`, requires creating a recurring event titled "Review session for Budget Planning," starting on 2023-10-15 at 14:00, lasting 60 minutes, repeating weekly without end, and including the description: "We will understand business objectives. Remember to confirm attendance."

In the initial steps, all agents successfully configured the title and basic settings. Their performance diverged in subsequent steps. The SFT agent made timing errors (setting 16:00 instead of 15:00) and executed redundant checks, revealing weak task understanding (Figure 8). The MobileRL w/o DGRPO agent skipped an adjustment step, yielding an incorrect event duration (Figure 9). By

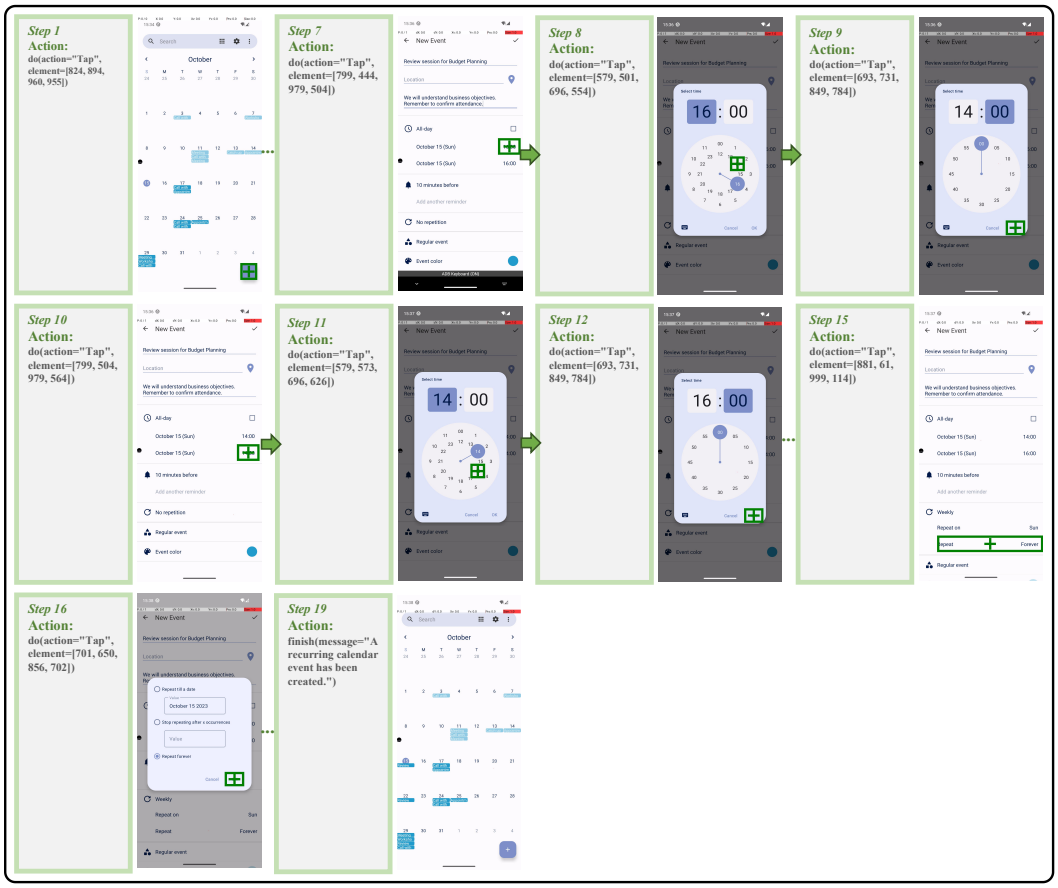


Figure 8: The model trained with expert data SFT, where the expert data does not include the reasoning part, in the `SimpleCalendarAddRepeatingEvent` task in AndroidWorld. Because of space constraints, we only present the most critical and notable steps for comparison.

contrast, the full MOBILERL agent completed the task accurately and efficiently, satisfying all requirements without redundant operations (Figure 10).

B HYPERPARAMETERS

Main hyperparameters are listed in Table 4.

C REWARD MODEL FOR ANDROIDLAB

C.1 METHOD OVERVIEW

Since the AndroidLab environment does not provide rule-based rewards for training data, we adopt a VLM-based reward model to supply reward signals during reinforcement learning. Specifically, we first execute all training and test tasks multiple times using medium-capability VLMs, including different versions of our models and proprietary VLMs, generating execution traces. Then, strong proprietary VLMs assign binary scores to each trace. For scoring, each model receives the full task screenshot sequence concatenated into a single image, along with step-by-step action descriptions, and is instructed to produce a reasoning process before outputting a score. For training-set tasks (non-rule-based), we use the majority vote among the three scores as the label and retain the reasoning; for test-set tasks, we use the existing rule-based reward as the label and keep only the model scores and reasoning consistent with it. This process yields several thousand labeled samples, which we use to fine-tune GLM-4.1V-9B-Thinking as the base model. After training, we evaluate the reward model

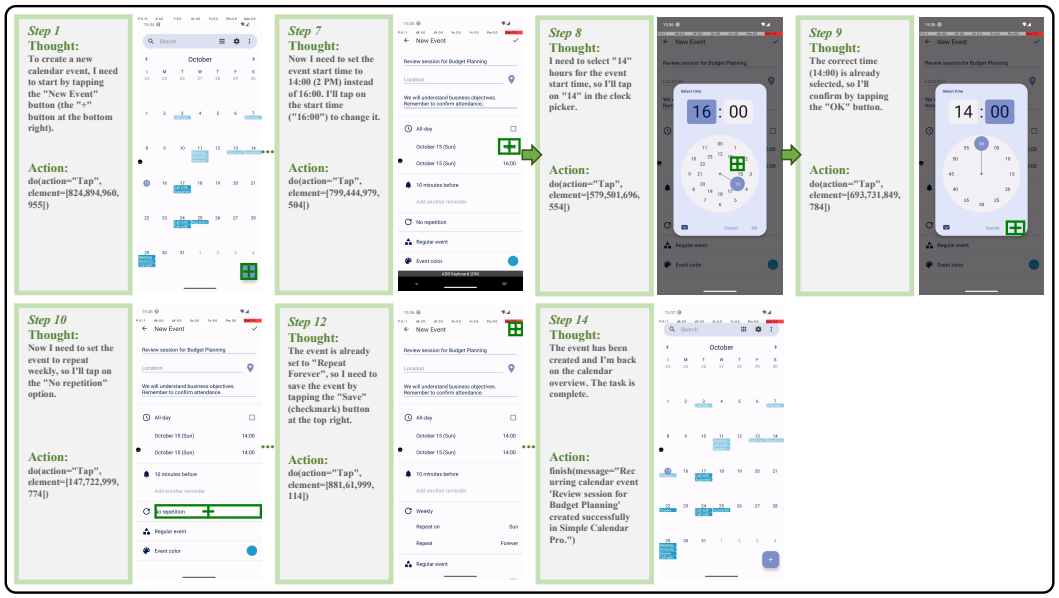


Figure 9: MOBILERL w/o DGRPO in the `SimpleCalendarAddRepeatingEvent` task in AndroidWorld. Because of space constraints, we only present the most critical and notable steps for comparison.

Table 4: Summary of Main Hyperparameters

Component	Hyperparameter	Value
Data	Max Prompt Length	16384
Data	Max Response Length	4096
Data	Train Batch Size	256
Data	Validation Batch Size	256
Actor / Policy	Strategy (Parallelism)	FSDP
Actor / Policy	PPO Micro Batch Size/GPU	4
Actor / Policy	Learning Rate (LR)	1e-6
Actor / Policy	Gradient Clipping	1.0
Actor / Policy	Clip Ratio	0.2
Actor / Policy	PPO Epochs	1
Rollout & Sampling	Sampling Temperature	1.0
Rollout & Sampling	Max New Tokens	4096
Rollout & Sampling	Number of Samples (n)	16
Rollout & Sampling	Max Turns	50
Rollout & Sampling	Max Pixels	500000
Rollout & Sampling	Min Pixels	65536
Algorithm	KL Loss Coefficient/ β	0.001
Algorithm	SPA/ α	1.0
Algorithm	DAPR/ Replay Buffer Size	256
Algorithm	DAPR/ γ	1.0
Algorithm	DAPR/ κ	0.25

on a curated set of 1000 AndroidLab traces with verified labels, selecting the best-performing version (86% accuracy) for online reinforcement learning.

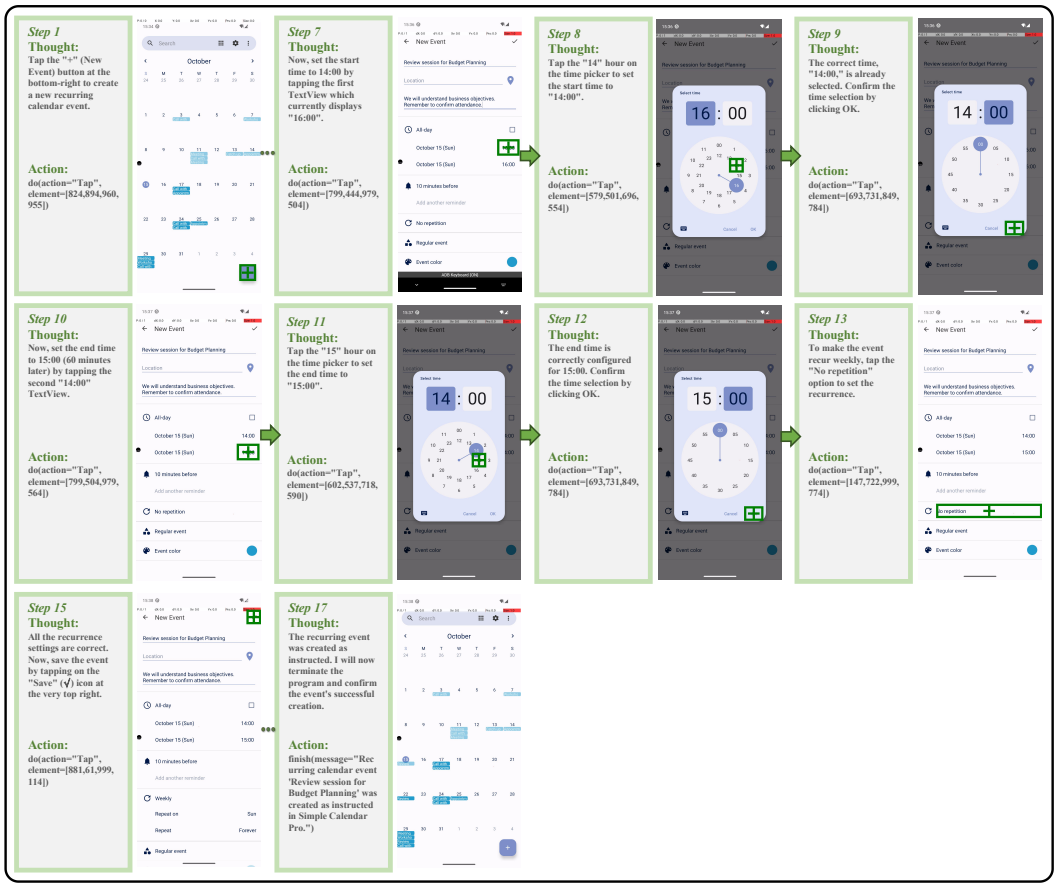


Figure 10: MOBILERL in the `SimpleCalendarAddRepeatingEvent` task in AndroidWorld. Because of space constraints, we only present the most critical and notable steps for comparison.

C.2 SYSTEM PROMPT FOR TRACE EVALUATION

The following system prompt guides the VLM in determining whether an agent has successfully completed a task.

```
You are an expert in determining whether a task
has been successfully and completely completed. You will receive:

1. The task description.
2. Step-by-step page states in XML format.
3. The agent's action descriptions.
4. A single image containing screenshots of all steps.
   - Green rectangles and crosses mark tap regions and positions.
   - Green arrows indicate swipe directions.
   - Red text shows typed input.

Action formats and meanings:
- do(action="Tap", element=[x1,y1,x2,y2])
    Tap on the specified screen region; green cross is the tap point.
- do(action="Launch", app="xxx")
    Launch the specified app.
- do(action="Type", text="xxx")
    Enter the specified text (shown in red).
- do(action="Swipe", element=[x1,y1,x2,y2], direction="x", dist="x")
```

```
    Swipe in the indicated direction; green arrow shows the swipe path.
- do(action="Long_Pres", element=[x1,y1,x2,y2])
    Long press on the specified region.
- do(action="Back")
    Navigate back to the previous screen.
- finish(message="xxx")
    End the task with the given message.

Scoring rules:
If the task is fully and correctly completed,
output a score of 1; otherwise, 0.

Output format:
<analysis>
Step 1 analysis: <Your analysis>
Step 2 analysis: <Your analysis>
...
Final step analysis: <Your analysis>
</analysis>
<ans>
[Your score]
</ans>
```

D XML PREPROCESSING FOR UI REPRESENTATION

The original XML from the Android accessibility service defines the layout and elements of the user interface, including all components on a page. As a result, it contains many nodes used solely for structural or layout purposes, which do not provide useful semantic information. Moreover, scrollable pages often contain more nodes than are visible on the screen, leading to the inclusion of many off-screen nodes.

D.1 REMOVAL OF OFF-SCREEN NODES

We first determine whether to retain off-screen nodes via the input parameter `remain_nodes`:

- **`remain_nodes=True`**: Off-screen nodes are preserved, e.g., when summarizing the full page content without requiring scrolling.
- **`remain_nodes=False`**: Off-screen nodes are removed to avoid interference during action simulation (e.g., tapping, scrolling).

In the original XML, a node is considered on-screen if its `bounds` property lies entirely within the screen dimensions $[0, 0]$ to $[Window\_Height, Window\_Width]$ and is contained by its parent node. We check this condition recursively to identify on-screen nodes.

D.2 REMOVAL OF REDUNDANT NODES

Nodes that do not convey functional or semantic information are removed. A node is considered *functional* if it satisfies at least one of the following:

- Any of the boolean attributes is `True`: `checkable`, `checked`, `clickable`, `focusable`, `scrollable`, `long-clickable`, `password`, `selected`.
- The `text` or `content-desc` attribute is non-empty.

All nodes failing these criteria are considered redundant and are deleted.

D.3 Attribute Simplification

Attribute descriptions in the original XML are verbose and token-expensive. We simplify them as follows:

- Keep only `True` values for the boolean functional attributes listed above (omit `False` values).
- Remove `index`, `resource-id`, and `package` (not useful for semantic understanding).
- For `class`, retain only the last component (e.g., `android.widget.FrameLayout` → `FrameLayout`).
- Merge `text` and `content-desc` attributes and display them separately.
- Retain `bounds` in full, as it indicates the node's position on the page.

D.4 Example Transformation

Original node:

```
<node index="0" text="Audio Recorder"
      resource-id="com.dimowner.audiorecorder:id/txt_title"
      class="android.widget.TextView" package="com.dimowner.audiorecorder"
      content-desc="" checkable="false" checked="false" clickable="false"
      enabled="true" focusable="false" focused="false" scrollable="false"
      long-clickable="false" password="false" selected="false"
      bounds="[221,1095][858,1222]" />
```

Simplified node:

```
TextView;;Audio Recorder;[221,1095][858,1222]
```